Evaluation of Cache-based Superscalar and Cacheless Vector Architectures for Scientific Computations

Leonid Oliker, Jonathan Carter, John Shalf, David Skinner
CRD/NERSC, Lawrence Berkeley National Laboratory, Berkeley, CA 94720

Stephane Ethier
Princeton Plasma Physics Laboratory, Princeton University, Princeton, NJ 08453

Rupak Biswas, Jahed Djomehri*, and Rob Van der Wijngaart*
NAS Division, NASA Ames Research Center, Moffett Field, CA 94035

Abstract

The growing gap between sustained and peak performance for scientific applications has become a well-known problem in high performance computing. The recent development of parallel vector systems offers the potential to bridge this gap for a significant number of computational science codes and deliver a substantial increase in computing capabilities. This paper examines the intranode performance of the NEC SX6 vector processor and the cache-based IBM Power3/4 superscalar architectures across a number of key scientific computing areas. First, we present the performance of a microbenchmark suite that examines a full spectrum of low-level machine characteristics. Next, we study the behavior of the NAS Parallel Benchmarks. Finally, we evaluate the performance of several numerical codes from key scientific computing domains. Overall results demonstrate that the SX6 achieves high performance on a large fraction of our application suite and in many cases significantly outperforms the cache-based architectures. However, certain classes of applications are not easily amenable to vectorization and would likely require extensive reengineering of both algorithm and implementation to utilize the SX6 effectively.

1 Introduction

The rapidly increasing peak performance and generality of superscalar cache-based microprocessors long led researchers to believe that vector architectures hold little promise for future large-scale computing systems. Due to their cost effectiveness, an ever-growing fraction of today's supercomputers employ commodity superscalar processors, arranged as systems of interconnected SMP nodes. However, the growing gap between sustained and peak performance for scientific applications on such platforms has become well known in high performance computing.

The recent development of parallel vector systems offers the potential to bridge this performance gap for a significant number of scientific codes, and to increase computational power substantially. This was highlighted dramatically when the Japanese Earth Simulator System's [2] results were published [14, 15, 18]. The Earth Simulator, based on NEC SX6[1] *vector technology*, achieves five times the LINPACK performance with half the number of processors of the IBM SP-based ASCI White, the world's fourth-most powerful supercomputer, built using *superscalar technology* [7]. In order to quantify what this new capability entails for scientific communities that rely on modeling and simulation, it is critical to evaluate these two microarchitectural approaches in the context of demanding computational algorithms.

In this paper, we compare the performance of the NEC SX6 vector processor against the cache-based IBM Power3 and Power4 architectures for several key scientific computing areas. We begin by evaluating memory bandwidth and MPI communication speeds, using a set of microbenchmarks. Next, we evaluate three of the well-known NAS Parallel Benchmarks (NPB) [4, 11], using problem size Class B. Finally, we present performance results for several numerical codes from scientific computing domains, including astrophysics, plasma fusion, fluid dynamics, magnetic fusion, and molecular dynamics. Since most modern scientific codes are already tuned for cache-based systems, we examine the effort required to port these applications to the vector architecture. We focus on serial and intranode parallel performance of our application suite, while isolating processor and memory behavior. Future work will explore the behavior of multi-node vector configurations.

*Employee of Computer Sciences Corporation.

[1] Also referred to as the Cray SX6 due to Cray's agreement to market NEC's SX line.

2 Architectural Specifications

We briefly describe the salient features of the three parallel architectures examined. Table 1 presents a summary of their intranode performance characteristics. Notice that the NEC SX6 has significantly higher peak performance, with a memory subsystem that features a data rate an order of magnitude higher than the IBM Power3/4 systems.

Node Type	CPU/ Node	Clock (MHz)	Peak (Gflops/s)	Memory BW (GB/s)	Peak Bytes/Flop	MPI Latency (μsec)
Power3	16	375	1.5	0.7	0.45	8.6
Power4	32	1300	5.2	2.3	0.44	3.0
SX6	8	500	8.0	32	4.0	2.1

Table 1: Architectural specifications of the Power3, Power4, and SX6 nodes.

2.1 Power3

The IBM Power3 was first introduced in 1998 as part of the RS/6000 series. Each 375 MHz processor contains two floating-point units (FPUs) that can issue a multiply-add (MADD) per cycle for a peak performance of 1.5 GFlops/s. The Power3 has a short pipeline of only three cycles, resulting in relatively low penalty for mispredicted branches. The out-of-order architecture uses prefetching to reduce pipeline stalls due to cache misses. The CPU has a 32KB instruction cache and a 128KB 128-way set associative L1 data cache, as well as an 8MB four-way set associative L2 cache with its own private bus. Each SMP node consists of 16 processors connected to main memory via a crossbar. Multi-node configurations are networked via the IBM Colony switch using an omega-type topology.

The Power3 experiments reported in this paper were conducted on a single Nighthawk II node of the 208-node IBM pSeries system (named Seaborg) running AIX 5.1 and located at Lawrence Berkeley National Laboratory.

2.2 Power4

The pSeries 690 is the latest generation of IBM's RS/6000 series. Each 32-way SMP consists of 16 Power4 chips (organized as 4 MCMs), where a chip contains two 1.3 GHz processor cores. Each core has two FPUs capable of a fused MADD per cycle, for a peak performance of 5.2 Gflops/s. Two load-store units, each capable of independent address generation, feed the two double precision MADDers. The superscalar out-of-order architecture can exploit instruction level parallelism through its eight execution units. Up to eight instructions can be issued each cycle into a pipeline structure capable of simultaneously supporting more than 200 instructions. Advanced branch prediction hardware minimizes the effects of the relatively long pipeline (six cycles) necessitated by the high frequency design.

Each processor contains its own private L1 cache (64KB instruction and 32KB data) with prefetch hardware; however, both cores share a 1.5MB unified L2 cache. Certain data access patterns may therefore cause L2 cache conflicts between the two processing units. The directory for the L3 cache is located on-chip, but the memory itself resides off-chip. The L3 is designed as a standalone 32MB cache, or to be combined with other L3s on the same MCM to create a larger interleaved cache of up to 128MB. Multi-node Power4 configurations are currently available employing IBM's Colony interconnect, but future large-scale systems will use the lower latency Federation switch.

The Power4 experiments reported here were performed on a single node of the 27-node IBM pSeries 690 system (named Cheetah) running AIX 5.1 and operated by Oak Ridge National Laboratory. The exception was the OVERFLOW-D code, which was run on a 64-processor system at NASA Ames Research Center.

2.3 SX6

The NEC SX6 vector processor uses a dramatically different architectural approach than conventional cache-based systems. Vectorization exploits regularities in the computational structure to expedite uniform operations on independent data sets. Vector arithmetic instructions involve identical operations on the elements of vector operands located in the vector register file. Many scientific codes allow vectorization, since they are characterized by predictable fine-grain data-parallelism that can be exploited with properly structured program semantics and sophisticated compilers. The 500 MHz SX6 processor contains an 8-way replicated vector pipe capable of issuing a MADD each cycle, for a peak performance of 8 Gflops/s per CPU. The processors contain 72 vector registers, each holding 256 64-bit words.

For non-vectorizable instructions, the SX6 contains a 500 MHz scalar processor with a 64KB instruction cache, a 64KB data cache, and 128 general-purpose registers. The 4-way superscalar unit has a peak of 1 Gflops/s and supports branch prediction, data prefetching, and out-of-order execution. Since the SX6 vector unit is significantly more powerful than the scalar processor, it is critical to achieve high vector operation ratios, either via compiler discovery or explicitly through code (re-)organization.

Unlike conventional architectures, the SX6 vector unit lacks data caches. Instead of relying on data locality to reduce memory overhead, memory latencies are masked by overlapping pipelined vector operations with memory fetches. The SX6 uses high speed SDRAM with peak bandwidth of 32GB/s per CPU: enough to feed one operand per cycle to each of the replicated pipe sets. Each SMP contains eight processors that share the node's memory. The nodes can be used as building blocks of large-scale multiprocessor systems; for instance, the Earth Simulator contains 640 SX6 nodes, connected through a single-stage crossbar.

The vector results in this paper were obtained on the single-node (8-way) SX6 system (named Rime) running SUPER-UX at the Arctic Region Supercomputing Center (ARSC) of the University of Alaska.

3 Microbenchmarks

This section presents the performance of a microbenchmark suite that measures some low-level machine characteristics such as memory subsystem behavior and scatter/gather hardware support (using STREAM [6]); and point-to-point communication, network/memory contention, and barrier synchronizations (using PMB [5]).

3.1 Memory Access Performance

First we examine the low-level memory characteristics of the three architectures in our study. Table 2 presents unit-stride memory bandwidth behavior of the triad summation: $a(i) = b(i) + s \times c(i)$, using the STREAM benchmark [6]. It effectively captures the peak bandwidth of the architectures, and shows that the SX6 achieves about 48 and 14 times the performance of the Power3 and Power4, respectively, on a single processor. Notice also that the SX6 shows negligible bandwidth degradation for up to eight tasks, while the Power3/4 drop by almost 50% for fully packed nodes.

Our next experiment concerns the speed of strided data access on a single processor. Figure 1 presents our results for the same triad summation, but using various memory strides. Once again, the SX6 achieves good bandwidth, up to two (three) orders of magnitude better than the Power4 (Power3), while showing markedly less average variation across the range of strides studied. Observe that certain strides impact SX6 memory bandwidth quite pronouncedly, by an order of magnitude or more. Analysis shows that strides containing factors of two worsen performance due to increased DRAM bank conflicts. On the Power3/4, a precipitous drop in data transfer rate occurs for small strides, due to loss of cache reuse. This drop is more complex on the Power4, because of its more complicated cache structure.

Finally, Figure 2 presents the memory bandwidth of indirect addressing through vector triad gather and scatter operations of various data sizes. For smaller sizes, the cache-based architectures show better data rates for indirect access

P	Power3	Power4	SX6
1	661	2292	31900
2	661	2264	31830
4	644	2151	31875
8	568	1946	31467
16	381	1552	—
32	—	1040	—

Table 2: Single-processor STREAM triad performance (in MB/s) for unit stride.

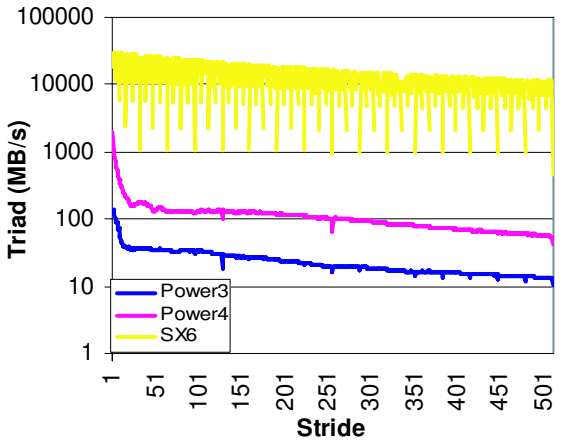

Figure 1: Single-processor STREAM triad performance (in MB/s) using regularly strided data.

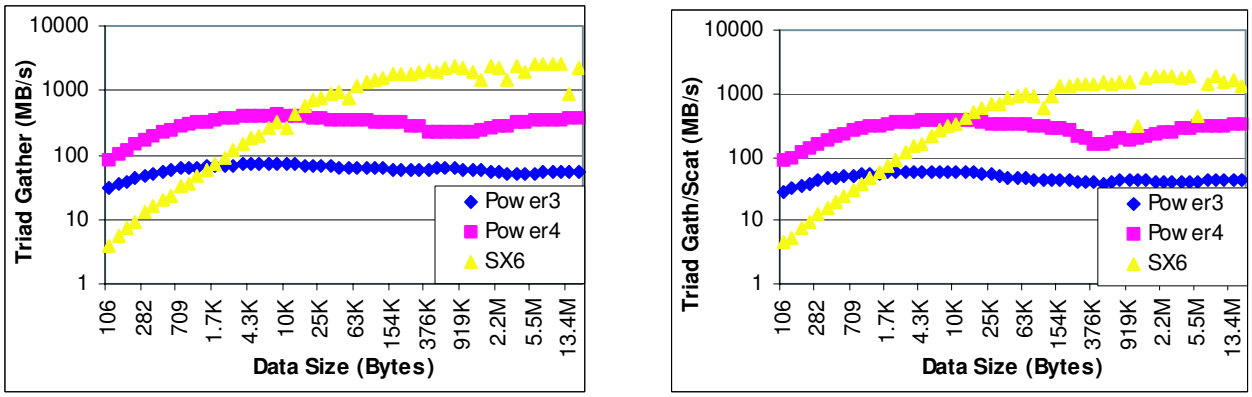

Figure 2: Single-processor STREAM triad performance (in MB/s) using irregularly strided data on right hand side (gather), and on left hand side as well as right hand side (gather/scatter).

to memory. However, for larger sizes, the SX6 is able to utilize its hardware gather and scatter support effectively, outperforming the cache-based systems.

3.2 MPI Performance

Message passing is the most wide-spread programming paradigm for high-performance parallel systems. The MPI library has become the de facto standard for message passing. It allows intra- and inter-node communications, thus obviating the need for hybrid programming schemes for distributed-memory systems. Although MPI increases code complexity compared with shared-memory programming paradigms such as OpenMP, its benefits lie in improved control over data locality and fewer global synchronizations.

Table 3 presents bandwidth figures obtained using the Pallas MPI Benchmark (PMB) suite [5], for exchanging intranode messages of various sizes. The first row shows the best case scenario when only two processors within a node communicate. Notice that the SX6 has significantly better performance, achieving more than 19 (7) times the bandwidth of the Power3 (Power4), for the largest messages. The effects of network/memory contention are visible when all processors within each SMP are involved in exchanging messages. Once again, the SX6 dramatically outperforms the Power3/4 architectures. For example, a message containing 524288 (2^{19}) bytes, suffers 46% (68%) bandwidth degradation when fully saturating the Power3 (Power4), but only 7% on the SX6.

	8192 Bytes			131072 Bytes			524288 Bytes			2097152 Bytes		
P	Power3	Power4	SX6	Power3	Power4	SX6	Power3	Power4	SX6	Power3	Power4	SX6
2	143	515	1578	408	1760	6211	508	1863	8266	496	1317	9580
4	135	475	1653	381	1684	6232	442	1772	8190	501	1239	9521
8	132	473	1588	343	1626	5981	403	1638	7685	381	1123	8753
16	123	469	—	255	1474	—	276	1300	—	246	892	—
32	—	441	—	—	868	—	—	592	—	—	565	—

Table 3: MPI send/receive performance (in MB/s) for various message sizes and processor counts.

Table 4 shows the overhead of MPI barrier synchronization (in μsec). As expected, the barrier overhead on all three architectures increases with the number of processors. For the fully loaded SMP test case, the SX6 has 3.6 (1.9) times lower barrier cost than Power3 (Power4); however, for the eight-processor test case, the SX6 performance degrades precipitously and is slightly exceeded by the Power4.

P	Power3	Power4	SX6
2	17.1	6.7	5.0
4	31.7	12.1	7.1
8	54.4	19.8	22.0
16	79.1	28.9	—
32	—	42.4	—

Table 4: MPI synchronization overhead (in μsec).

4 Scientific Kernels: NPB

The NAS Parallel Benchmarks (NPB) [4, 11] provide a good middle ground for evaluating the performance of compact, well-understood applications. The NPB were created at a time when vector machines were considered no longer cost effective, and although their performance was meant to be good across a whole range of systems, they were written with cache-based systems in mind. Here we investigate the work involved in producing good NPB vector code. Of the eight published NPB, we select the three most appropriate for our current study: CG, a sparse-matrix conjugate-gradient algorithm marked by irregular stride resulting from indirect addressing; FT, an FFT kernel; BT, a synthetic flow solver that features simple recurrences in a different array index in three different parts of the solution process. In Table 5 we present MPI performance results for these codes on the SX6 and Power3 for medium problem sizes, commonly referred to as Class B. Performance results are reported in Aggregate Mflops/s (AM). To characterize vectorization behavior we also show *average vector length* (AVL) and *vector operation ratio* (VOR). Cache effects are accounted for by TLB misses in % per cycle (TLB) and L1 hits in % per cycle (L1).

	Power3									SX6								
	CG			FT			BT			CG			FT			BT		
P	AM	L1	TLB	AM	L1	TLB	AM	L1	TLB	AM	AVL	VOR	AM	AVL	VOR	AM	AVL	VOR
1	53.76	68.0	.058	133.4	91.1	.204	144.3	96.9	.039	469.5	198.6	96.9	2021.	256.0	98.4	3693.	100.9	99.2
2	110.9	71.9	.040	239.2	91.2	.088	—	—	—	517.0	147.0	96.0	2691.	255.7	98.4	—	—	—
4	215.9	73.4	.027	467.9	91.6	.087	509.8	97.2	.031	1013.	147.9	96.5	5295.	255.2	98.4	9581.	51.2	98.7
8	438.3	80.7	.032	897.6	91.6	.084	—	—	—	1045.	117.1	95.0	9933.	254.0	98.4	—	—	—
9	—	—	—	—	—	—	1097.	97.4	.032									
16	765.3	85.6	.030	1519.	91.6	.070	1646.	97.4	.025									

Table 5: NAS Parallel Benchmarks Class B performance results. AVL is Average Vector Length, VOR is Vector Operation Ratio, and AM is Aggregate MFlops/s.

Although the CG code vectorizes fully and exhibits fairly long vector lengths, uni-processor SX6 performance is not very good due to many bank conflicts resulting from the indirect addressing. Multi-processor SX6 speedup degrades as expected with the reduction in vector length. Power3 scalability is very good, mostly because uni-processor performance is so poor due to the serious lack of data locality. FT did not perform well on the SX6 in its original form, because the computations used a fixed block length of 16 words. But once the code was modified to use a block length equal to the size of the grid (only three lines changed), SX6 uni-processor performance improved markedly due to increased vector length. Speedup from one to two processors was not good due to the time spent in a routine that does a local data transposition to improve data locality for cache based machines (this routine was not called in the uni-processor run), but subsequent scalability was excellent. Power3 scalability was fairly good overall, despite the large communication volume, due to improved data locality of the multi-processor implementation. The BT baseline MPI code performed poorly on the SX6, because subroutines in inner loops inhibited vectorization. Also, some inner loops of small fixed length were vectorized, leading to very short vector lengths. Subroutine inlining and manual expansion of small loops leads to long vector lengths throughout the single-processor code, and good performance. Increasing the number of processors on the SX6 causes reduction of vector length (artifact of the three-dimensional domain decomposition) and a concomitant deterioration of the speedup. Power3 scalability is fair up to 9 processors, but degrades severely on 16 processors. The reason is the fairly large number of synchronizations per time step that are costly on a fully saturated 16-processor Power3 node. Experiments with a two-node computation involving 25 processors show a remarkable recovery of the speedup.

5 Application Performance Metrics

Five applications from diverse areas in scientific computing were chosen to measure and compare the performance of the SX6 with that of the Power3 and Power4. The applications are: Cactus, an astrophysics code that solves Einstein's equations; TLBE, a fusion energy application that performs simulations of high-temperature plasma; OVERFLOW-D, a CFD production code that solves the Navier-Stokes equations around complex aerospace configurations; GTC, a particle-in-cell approach to solve the gyrokinetic Vlasov-Poisson equations; and Mindy, a simplified molecular dynamics code that uses the Particle Mesh Ewald algorithm. Performance results are reported in Mflops/s per processor, except where the original algorithm has been modified for the SX6 (these are reported as wall-clock time). Flops are obtained with the `hpmcount` tool on the Power3/4 and `ftrace` on the SX6.

6 Astrophysics: Cactus

One of the most challenging problems in astrophysics is the numerical solution of Einstein's equations following from the Theory of General Relativity (GR): a set of coupled nonlinear hyperbolic and elliptic equations containing thousands of terms when fully expanded. The Albert Einstein Institute in Potsdam, Germany, developed the Cactus code [1] to evolve these equations stably in 3D on supercomputers to simulate astrophysical phenomena with high gravitational fluxes, such as the collision of two black holes and the gravitational waves that radiate from that event.

6.1 Methodology

The core of the Cactus solver uses the ADM formalism, also known as the 3+1 form. In GR, space and time form a 4D space (three spatial and one temporal dimension) that can be sliced along any dimension. For the purpose of solving Einstein's equations, the ADM solver decomposes the solution into 3D spatial hypersurfaces that represent different slices of space along the time dimension. In this formalism, the equations are written as four constraint equations and 12 evolution equations. The evolution equations can be solved using a number of different numerical methods, including staggered leapfrog, McCormack, Lax-Wendroff, and iterative Crank-Nicholson schemes. A "lapse" function describes the time slicing between hypersurfaces for each step in the evolution. A "shift metric" is used to move the coordinate system at each step to avoid being drawn into a singularity. The four constraint equations are used to select different lapse functions and the related shift vectors.

For performance evaluation we focused on a core Cactus ADM solver, the Fortran77-based ADM kernel (BenchADM [8]), written when vector machines were more common; consequently, we expect it to vectorize well. BenchADM is computationally intensive, involving 600 flops per grid point. The loop body of the most numerically intensive part of the solver is large (several hundred lines of code). Splitting this loop provided little or no performance enhancement, as expected, due to little register pressure in the default implementation.

6.2 Porting Details

BenchADM vectorized almost entirely on the SX6 in the first attempt. However, the vectorization appears to involve only the innermost of a triply nested loop (x, y, and z-directions for a 3D evolution). The resulting effective vector length for the code is directly related to the x-extent of the computational grid.

6.3 Performance Results

Table 6 presents performance results for BenchADM on a 127^3 grid. The mild deterioration of the performance on the Power3/4 as the number of processors grows up to 16 is due to the cost of communications, with a steep drop in performance as a Power4 node gets fully saturated (32 processors). Increasing the grid size to 128^3 results in severe performance degradation, even though TLB miss and L1 hit rates are hardly affected. Apparently, that degradation is attributable primarily to poor L2 and L3 cache reuse.

While for smaller grid sizes the SX6 performance is mediocre, the 127^3 grid uni-processor computation returns an impressive 3.9 GFlops/s with a sizable AVL and a VOR of almost 100%. To date, SX6's 50% of peak performance is the best achieved for this benchmark on any current computer architecture. SX6 multi-processor performance deteriorates fairly rapidly due to the rising cost of inter-processor synchronization (see Table 4); the AVL and VOR

are hardly affected by the parallelization, and artificially changing the volume of communications has negligible effect on performance.

	Power3			Power4			SX6		
P	Mflops/s	L1	TLB	Mflops/s	L1	TLB	Mflops/s	AVL	VOR
1	273.8	99.4	.030	672.1	92.2	.01	3912.	126.7	99.6
2	236.2	99.4	.030	582.4	92.6	.01	3500.	126.7	99.5
4	248.9	99.4	.020	618.8	93.2	.01	2555.	126.7	99.5
8	251.4	99.4	.030	599.6	92.4	.01	2088.	126.7	99.3
16	226.5	99.5	.020	537.6	93.0	.01			
32	—	—	—	379.4	97.0	.00			

Table 6: Performance of the Cactus BenchADM kernel on a 127^3 grid.

7 Plasma Fusion: TLBE

Lattice Boltzmann methods provide a mesoscopic description of the transport properties of physical systems using a linearized Boltzmann equation. They offer an efficient way to model turbulence and collisions in a fluid. The TLBE application [16] performs a 2D simulation of high-temperature plasma using a hexagonal lattice and the BGK collision operator.

7.1 Methodology

The TLBE simulation has three computationally demanding components: computation of the mean macroscopic variables (integration); relaxation of the macroscopic variables after colliding (collision); and propagation of the macroscopic variables to neighboring grid points (stream). The first two steps are floating-point intensive, the third consists of data movement only. The problem is ideally suited for vector architectures. The first two steps are completely vectorizable, since the computation for each grid point is purely local. The third step consists of a set of strided copy operations. In addition, distributing the grid via a 2D decomposition easily parallelizes the method. The first two steps require no communication, while the third has a regular, static communication pattern in which the boundary values of the macroscopic variables are exchanged.

7.2 Porting Details

After initial profiling on the SX6 using basic vectorization compiler options (`-C vopt`), a poor result of 280 Mflops/s was achieved for a small 64^2 grid using a serial version of the code. `ftrace` showed that VOR was high (95%) and that the collision step dominated the execution time (96% of total); however, AVL was only about 6. We found that the inner loop over the number of directions in the hexagonal lattice had been vectorized, but not a loop over one of the grid dimensions. Invoking the most aggressive compiler flag (`-C hopt`) did not help. Therefore, we rewrote the collision routine by creating temporary vectors, and inverted the order of two loops to ensure vectorization over one dimension of the grid. As a result, serial performance improved by a factor of 7, and the parallel TLBE version was created by inserting the new collision routine into the MPI version of the code.

7.3 Performance Results

Parallel TLBE performance using a production grid of 2048^2 is presented in Table 7. The SX6 results show that TLBE achieves almost perfect vectorization in terms of AVL and VOR. The 2- and 4-processor runs show similar performance as the serial version; however, an appreciable degradation is observed when running 8 MPI tasks, which is most likely due to network/memory contention in the SMP.

For both the Power3 and Power4 architectures, the collision routine rewritten for the SX6 performed somewhat better than the original. On the cache-based machines, the parallel TLBE showed higher Mflops/s (per CPU) compared with the serial version. This is due to the use of smaller grids per processor in the parallel case, resulting in improved cache reuse. The more complex behavior on the Power4 is due to the competitive effects of the three-level cache

P	Power3			Power4			SX6		
	Mflops/s	L1	TLB	Mflops/s	L1	TLB	Mflops/s	AVL	VOR
1	70	90.5	.50	250	58.2	.069	4060	256	99.5
2	110	91.7	.77	300	69.2	.014	4060	256	99.5
4	110	91.7	.75	310	71.7	.013	3920	256	99.5
8	110	92.4	.77	470	87.1	.021	3050	255	99.2
16	110	92.6	.73	460	88.7	.019			
32	—	—	—	440	89.3	.076			

Table 7: Performance of TLBE on a 2048^2 grid.

structure and saturation of the SMP memory bandwidth. In summary, using all 8 CPUs on the SX6 gives an aggregate performance of 24.4 Gflops/s (38% of peak), and a speedup factor of 27.7 (6.5) over the Power3 (Power4), with minimal porting overhead.

8 Fluid Dynamics: OVERFLOW-D

OVERFLOW-D [17] is an overset grid methodology [9] for high-fidelity Navier-Stokes CFD simulations around complex aerospace configurations. The application can handle complex designs with multiple geometric components, where individual body-fitted grids are easily constructed about each component. OVERFLOW-D is designed to simplify the modeling of components in relative motion (dynamic grid systems). At each time step, the flow equations are solved independently on each grid ("block") in a sequential manner. Boundary values in grid overlap regions are updated before each time step, using a Chimera interpolation procedure. The code uses finite differences in space, and implicit/explicit time stepping.

8.1 Methodology

The MPI version of OVERFLOW-D (in F90) is based on the multi-block feature of the sequential code, which offers natural coarse-grain parallelism. The sequential code consists of an outer "time-loop" and an inner "grid-loop". The inter-grid boundary updates in the serial version are performed successively. To facilitate parallel execution, grids are clustered into groups; one MPI process is then assigned to each group. The grid-loop in the parallel implementation contains two levels, a loop over groups ("group-loop") and a loop over the grids within each group. The group-loop is performed in parallel, with each group performing its own sequential grid-loop and inter-grid updates. The inter-grid boundary updates across the groups are achieved via MPI.

8.2 Porting Details

The MPI implementation of OVERFLOW-D is based on the sequential version, the organization of which was designed to exploit vector machines. The same basic code structure is used on both the Power4 and the SX6, except for the LU-SGS linear solver that required significant modifications to enhance efficiency. On the Power3/4, a pipeline [10] strategy was implemented, while on the SX6, a skewed hyper-plane algorithm was used. These changes were dictated by the data dependencies inherited by the solution process, and to take advantage of the cache and vector architectures, respectively. A few other minor changes were also made in some subroutines in an effort to meet specific compiler requirements.

8.3 Performance Results

Our experiments involve a Navier-Stokes simulation of vortex dynamics in the complex wake flow region around hovering rotors. The grid system consisted of 41 blocks and approximately 8 million grid points. Table 8 shows execution times per time step (averaged over 10 steps) on the Power3/4 and SX6. Results show that the SX6 outperforms the Power3/4; in fact, the run time for 8 processors on the SX6 is less than half the 32-processor Power4 number. Scalability is similar for both Power4 and SX6 architectures, with computational efficiency decreasing for a larger number of MPI tasks due largely to load imbalance. Power3 scalability exceeds that of Power4 and SX6. On the SX6, the

relatively small AVL and limited VOR explain why the code achieves a maximum of only 7.8 Gflops/s on 8 processors. Reorganizing OVERFLOW-D would achieve higher vector performance; however, extensive effort would be required to modify this production code.

	Power3			Power4	SX6		
P	sec	L1	TLB	sec	sec	AVL	VOR
2	46.7	93.3	.245	15.8	5.5	87	80
4	26.6	95.4	.233	8.5	2.8	84	76
8	13.2	96.6	.197	4.3	1.6	79	69
16	8.0	98.2	.143	3.7			
32	—	—	—	3.4			

Table 8: Performance of OVERFLOW-D on a 8 million-grid point problem

9 Magnetic Fusion: GTC

The goal of magnetic fusion is the construction and operation of a burning plasma power plant producing clean energy. The performance of such a device is determined by the rate at which the energy is transported out of the hot core to the colder edge of the plasma. The Gyrokinetic Toroidal Code (GTC) [13] was developed to study the dominant mechanism for this transport of thermal energy, namely plasma microturbulence. Plasma turbulence is best simulated by particle codes, in which all the nonlinearities are naturally included.

9.1 Methodology

GTC solves the gyroaveraged Vlasov-Poisson (gyrokinetic) system of equations [12]) using the particle-in-cell approach. Instead of interacting with each other, the simulated particles interact with a self-consistent electrostatic or electromagnetic field described on a grid. Numerically, the PIC method scales as N, instead of N^2 as in the case of direct binary interactions. Also, the equations of motion for the particles are simple ODEs (rather than nonlinear PDEs), and can be solved easily (e.g. Runge-Kutta). The main tasks at each time step are: deposit the charge of each particle at the nearest grid points (scatter); solve the Poisson equation to get the potential at each grid point; calculate the force acting on each particle from the potential at the nearest grid points (gather); move the particles by solving the equations of motion; find the particles that have moved outside their local domain and migrate them accordingly.

The parallel version of GTC performs well on massive superscalar systems, since the Poisson equation is solved as a local operation. The key performance bottleneck is the scatter operation, a loop over the array containing the position of each particle. Based on a particle's position, we find the nearest grid points surrounding it and assign each of them a fraction of its charge proportional to the separation distance. These charge fractions are then accumulated in another array. The scatter algorithm in GTC is complicated by the fact that these are fast gyrating particles, where motion is described by charged rings being tracked by their guiding center (the center of the circular motion).

9.2 Porting Details

GTC's scatter phase presented some challenges when porting the code to the SX6 architecture. It is difficult to implement efficiently due to its non-contiguous writes to memory. The particle array is accessed sequentially, but its entries correspond to random locations in the simulation space. As a result, the grid array accumulating the charges is accessed in random fashion, resulting in poor cache performance. This problem is exacerbated on vector architectures, since many particles deposit charges at the same grid point, causing a classic memory dependence problem and preventing vectorization. We avoid these memory conflicts by using temporary arrays of vector length (256 words) to accumulate the charges. Once the loop is completed, the information in the temporary array is merged with the real charge data; however, this increases memory traffic and reduces the flop/byte ratio.

Another source of performance degradation was a short inner loop located inside two large particle loops that the SX6 compiler could not vectorize. This problem was solved by inserting a vectorization directive, fusing the inner and outer loops. Finally, I/O within the main loop had to be removed in order to allow vectorization.

9.3 Performance Results

Table 9 shows GTC performance results. The simulation in this study comprises 4 million particles and 301,472 grid points. The geometry is a torus described by the configuration of the magnetic field. On a single processor the Power3 sustains 153 Mflops/s (10% of peak), while the 277 Mflops/s achieved on the Power4 represents only 5% of its peak performance. The SX6 single-processor experiment runs at 701 Mflops/s, or only 9% of its theoretical peak. This poor performance is unexpected, considering the relatively high AVL (180) and VOR (97%). We believe this is because the scalar units need to compute the indices for the scatter/gather of the underlying unstructured grid. However, the SX6 still outperforms the Power3/4 by factors of 4.6 and 2.5, respectively.

	Power3			Power4			SX6		
P	Mflops/s	L1	TLB	Mflops/s	L1	TLB	Mflops/s	AVL	VOR
1	153	95.1	.130	277	89.4	.015	701.4	186.8	98.0
2	155	95.1	.102	294	89.8	.009	652.7	184.8	98.0
4	163	96.0	.084	310	91.2	.007	547.5	181.5	97.9
8	167	97.3	.052	326	92.2	.006	390.8	175.4	97.7

Table 9: Performance of GTC on a 4-million particle simulation.

10 Molecular Dynamics: Mindy

Mindy is a simplified serial molecular dynamics (MD) C++ code, derived from the parallel MD program "NAMD" [3]. The energetics, time integration, and file formats are identical to those used by NAMD.

10.1 Methodology

Mindy's core is the calculation of forces between N atoms via the Particle Mesh Ewald (PME) algorithm. Its $O(N^2)$ complexity is reduced to $O(N \log N)$ by dividing the problem into boxes, and then computing electrostatic interaction in aggregate by considering neighboring boxes. Neighbor lists and a variety of cutoffs are used to decrease the required number of force computations.

10.2 Porting Details

Modern MD codes such as Mindy present special challenges for vectorization, since many optimization and scaling methodologies are at odds with the flow of data suitable for vector architectures. The reduction of floating point work from N^2 to $N \log N$ is accomplished at the cost of increased branch complexity and nonuniform data access. These techniques have a deleterious effect on vectorization; two strategies were therefore adopted to optimize Mindy on the SX6. The first severely decreased the number of conditions and exclusions in the inner loops, resulting in more computation overall, but less inner-loop branching. We refer to this strategy as NO_EXCL.

The second approach was to divide the electrostatic computation into two steps. First, the neighbor lists and distances are checked for exclusions, and a temporary list of inter-atom forces to be computed is generated. The force computations are then performed on this list in a vectorizable loop. Extra memory is required for the temporaries and, as a result, the flop/byte ratio is reduced. This scheme is labeled BUILD_TEMP.

Mindy uses C++ objects extensively, hindering the compiler to identify data-parallel code segments. Aggregate datatypes call member functions in the force computation, which impede vectorization. Compiler directives were used to specify that certain code sections contain no dependencies, allowing partial vectorization of those regions.

10.3 Performance Results

The case studied here is the apolipoprotein A-I, a 92224-atom system important in cardiac blood chemistry that has been adopted as a benchmark for large scale MD simulations on biological systems. Table 10 presents performance results of the serial Mindy algorithm. Neither of the two SX6 optimization strategies achieves high performance. The NO_EXCL approach results in a very small VOR, meaning that almost all the computations are performed on

the scalar unit. The BUILD_TEMP (also used on the Power3/4) approach increases VOR, but incurs the overhead of increased memory traffic for storing temporary arrays. In general, this class of applications is at odds with vectorization due to the irregularly structured nature of the codes. The SX6 achieves only 165 Mflops/s, or 2% of peak, slightly outperforming the Power3 and trailing the Power4 by about a factor of two in runtime. Effectively utilizing the SX6 would likely require extensive reengineering of both the algorithm and the code.

Power3			Power4			SX6: NO_EXCL			SX6: BUILD_TEMP		
sec	L1	TLB	sec	L1	TLB	sec	AVL	VOR	sec	AVL	VOR
15.7	99.8	0.01	7.8	98.8	.001	19.7	78	0.03	16.1	134	34.8

Table 10: Serial performance of Mindy on a 92224-atom system with two different SX6 optimization approaches.

11 Summary and Conclusions

This paper presented the performance of the NEC SX6 vector processor and compared it against the cache-based IBM Power3/4 superscalar architecture, across a range of scientific computations. Experiments with a set of microbenchmarks demonstrated that for low-level program characteristics, the specialized SX6 vector hardware significantly outperforms the commodity-based superscalar designs of the Power3 and Power4. Next we examined the NPBs, a well-understood set of kernels representing key areas in scientific computations. These compact codes allowed us to perform the three main variations of vectorization tuning: compiler flags, compiler directives, and actual code modifications. Results enabled us to identify classes of applications both at odds with and well suited for vector architectures, with performance ranging from 5% to 46% of peak on a single SX6 processor.

Several applications from key scientific computing domains were also evaluated. Table 11 summarizes the relative performance results. Since most modern scientific codes are designed for (super)scalar systems, we examined the effort required to port these applications to the vector architecture. Results show that the SX6 achieves decent performance for a large fraction of our application suite and in many cases significantly outperforms the scalar architectures. The computationally intensive Cactus BenchADM code showed the best uni-processor vector performance (46%), achieving a factor of 14.3 (5.8) improvement over the Power3 (Power4), while only requiring recompilation on the SX6.

Name	Discipline	Lines of Code	Power3 % Pk	Power4 %Pk	SX6 % Pk	P	SX6 Speedup vs. Power3	SX6 Speedup vs. Power4
Cactus-ADM	Astrophysics	1200	16.7	11.5	26.1	8	14	5.8
TLBE	Plasma Fusion	1500	7.3	9.0	38.1	8	27.8	6.5
OVERFLOW-D	Fluid Dynamics	100000	7.8	6.9	12.1	8	8.2	2.7
GTC	Magnetic Fusion	5000	11.1	6.3	4.9	8	2.3	1.2
Mindy	Molecular Dynamics	11900	6.3	4.7	2.1	1	1.0	0.5

Table 11: Summary overview of application suite performance

The rest of our applications required the insertion of compiler directives and/or minor code modifications to improve the two critical components of effective vectorization: long vector length and high vector operation ratio. Vector optimization strategies included loop fusion (and loop reordering) to improve vector length; introduction of temporary variables to break loop dependencies (both real and compiler imagined); reduction of conditional branches; and alternative algorithmic approaches. For codes such as TLBE, minor code changes were sufficient to achieve good vector performance and a high percentage of theoretical peak, especially for the multi-processor computations. For OVERFLOW-D, we obtained fair performance on both the cache-based and vector machines, but algorithmic support for these different architectures required substantial differences in programming styles.

Finally, we presented two applications with poor vector performance: GTC and Mindy. They feature indirect addressing, many conditional branches, and loop carried data-dependencies, making high vector performance challenging. This was especially true for Mindy, whose use of C++ objects made it difficult for the compiler to identify data-parallel loops. Effectively utilizing the SX6 would likely require extensive reengineering of both the algorithm and the implementation for these applications.

Acknowledgements

All authors from Lawrence Berkeley National Laboratory were supported by Director, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098. The Computer Sciences Corporation employees were supported by NASA Ames Research Center under contract number DTTS59-99-D-00437/A61812D with AMTI/CSC.

References

[1] Cactus Code Server. http://www.cactuscode.org.

[2] Earth Simulator Center. http://www.es.jamstec.go.jp.

[3] Mindy — A 'minimal' molecular dynamics program. http://www.ks.uiuc.edu/Development/MDTools/mindy.

[4] NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB.

[5] Pallas MPI Benchmarks. http://www.pallas.com/e/products/pmb.

[6] STREAM: Sustainable memory bandwidth in high performance computers. http://www.cs.virginia.edu/stream.

[7] Top500 Supercomputer Sites. http://www.top500.org.

[8] A. Abrahams, D. Bernstein, D. Hobill, E. Seidel, and L. Smarr. Numerically generated black hole spacetimes: Interaction with gravitational waves. *Phys. Rev. D*, 45:3544–3558, 1992.

[9] P.G. Buning, D.C. Jespersen, T.H. Pulliam, W.M. Chan, J.P. Slotnick, S.E. Krist, and K.J. Renze. *Overflow User's Manual, Version 1.8g*. NASA Langley Research Center, 1999.

[10] M.J. Djomehri and H. Jin. Hybrid MPI+OpenMP programming of an overset CFD solver and performance investigations. Technical Report NAS-02-002, NASA Ames Research Center, 2002.

[11] D.H. Bailey et al. The NAS Parallel Benchmarks. Technical Report RNR-94-007, NASA Ames Research Center, 1994.

[12] W.W. Lee. Gyrokinetic particle simulation model. *J. Comp. Phys.*, 72:243–262, 1987.

[13] Z. Lin, S. Ethier, T.S. Hahm, and W.M. Tang. Size scaling of turbulent transport in magnetically confined plasmas. *Phys. Rev. Lett.*, 88, 2002. 195004.

[14] H. Sakagami, H. Murai, Y. Seo, and M. Yokokawa. 14.9 TFLOPS three-dimensional fluid simulation for fusion science with HPF on the Earth Simulator. In *Proc. SC2002*, 2002.

[15] S. Shingu, H. Takahara, H. Fuchigami, M. Yamada, Y. Tsuda, W. Ohfuchi, Y. Sasaki, K. Kobayashi, T. Hagiwara, S. Habata, M. Yokokawa, H. Itoh, and K. Otsuka. A 26.58 Tflops global atmospheric simulation with the spectral transform method on the Earth Simulator. In *Proc. SC2002*, 2002.

[16] G. Vahala, J. Carter, D. Wah, L. Vahala, and P. Pavlo. *Parallelization and MPI performance of Thermal Lattice Boltzmann codes for fluid turbulence*. Parallel Computational Fluid Dynamics '99. Elsevier, 2000.

[17] A.M. Wissink and R. Meakin. Computational fluid dynamics with adaptive overset grids on parallel and distributed computer platforms. In *Proc. Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, pages 1628–1634, 1998.

[18] M. Yokokawa, K. Itakura, A. Uno, T. Ishihara, and Y. Kaneda. 16.4-Tflops Direct Numerical Simulation of turbulence by Fourier spectral method on the Earth Simulator. In *Proc. SC2002*, 2002.